ENGENHARIA DE SOFTWARE

AULA 4

Prof. Alex Mateus Porn

CONVERSA INICIAL

Olá! Nesta aula estudaremos as estimativas de esforço para desenvolvimento de softwares. Como esforço, consideraremos, nesta aula, o número de desenvolvedores/mês, para obtenção de valores aproximados de tamanho adequado de uma equipe de desenvolvimento, do tempo necessário até a entrega do projeto e do custo aproximado de um software.

Como nas aulas anteriores, abordamos os processos prescritivos e ágeis que compõem e gerenciam um projeto de desenvolvimento de softwares. Nesta aula, temos como objetivo principal realizar estimativas adequadas, para possibilitar que os modelos de processos e arquiteturas de software selecionados sejam aplicados em tempo hábil, com um tamanho de equipe adequado e, principalmente, com estimativa de um custo aproximado. Estudaremos cinco diferentes técnicas de estimativas, das mais antigas até as mais recentes, porém não em uma ordem cronológica, mas sim identificando as relações e derivações entre elas.

A aula será finalizada com a análise do *constructive cost model* (Cocomo), que é o principal modelo de estimativa de tamanho da equipe e tempo linear de desenvolvimento de softwares, ou seja, um modelo que auxilia a todas as demais técnicas nesses quesitos. Ao longo desta aula serão trabalhados os conteúdos da Figura 1.

Figura 1 – Temas e subtemas da aula

TEMA 1 – ANÁLISE DE PONTOS DE FUNÇÃO

Análise de ponto de função é uma técnica de medição do tamanho funcional de um software. Para Lopes (2011), com essa medição é possível estimar o esforço para implementação de um sistema. Essa técnica tem por definição medir o que o software faz e não como ele foi construído. Portanto, o processo de medição é fundamentado em uma avaliação padronizada dos requisitos lógicos do usuário.

Ainda parafraseando Lopes (2011), os pontos de função não medem diretamente o esforço, a produtividade, o custo ou outras informações específicas. É exclusivamente uma medida de tamanho funcional de software que, aliada à estimação de outras variáveis, poderá ser usada para derivar produtividade, custo e estimar esforço. A análise de pontos por função foi especificada por Allan J. Albrecht (1979), sendo transformada em uma metodologia formal em 1984.

De acordo com Silva e Oliveira (2005, p. 4):

> Um ponto de função deve ser entendido como uma unidade de medida que procura definir o tamanho de um aplicativo (software), independentemente de como possa ser produzido e implementado. A ideia central é que a medida funcional do software esteja respaldada nos requisitos lógicos dos usuários.

A Figura 2 apresenta o processo de contagem dos pontos de função de um aplicativo.

Figura 2 – Processo de contagem de pontos de função de um aplicativo

Fonte: Elaborado com base em Lopes, 2011, p. 6.

Conforme podemos observar na Figura 2, esse processo de contagem é dividido em seis etapas, sendo elas:

1. determinar o tipo de contagem;

2. identificar o escopo da contagem e a fronteira da aplicação;

3. contar funções:

a. tipo dados;

b. tipo transação;

4. determinar a contagem de pontos de função não ajustados;

5. determinar o valor do fator de ajuste;

6. calcular o número dos pontos de função ajustados (AFP).

1.1 DETERMINAR O TIPO DE CONTAGEM

A primeira fase do processo consiste em determinar o tipo de contagem do projeto de software, que, conforme a análise de pontos de função, pode ser de três tipos:

1. **Projeto de desenvolvimento**: contagem de pontos de função associados à criação de um novo projeto.

2. **Projeto de melhoria**: contagem de pontos de função de funções adicionadas, modificadas ou eliminadas de um projeto existente.

3. **Aplicação**: contagem de pontos de função de um projeto finalizado.

1.2 IDENTIFICAR O ESCOPO DE CONTAGEM E A FRONTEIRA DA APLICAÇÃO

Conforme abordado por Silva e Oliveira (2005, p. 5), a identificação do escopo de contagem refere-se a determinar se a contagem estará concentrada em um ou mais sistemas ou mesmo em partes de um ou mais sistemas. Já para Lopes (2011), esse escopo define quais funções serão incluídas na contagem, podendo abranger todas as funcionalidades, apenas as utilizadas ou algumas específicas.

Com relação à fronteira da aplicação, Silva e Oliveira (2005) apontam que ela estabelece um divisor entre os componentes do aplicativo e os componentes de outros aplicativos. Como complemento, Lopes (2011, p. 6) infere que a fronteira da aplicação a ser contada é a linha que separa uma aplicação de outra, de modo que em um mesmo escopo podem existir mais de uma aplicação a serem contadas, por isso a necessidade de definição da fronteira.

1.3 CONTAGEM DAS FUNÇÕES

A terceira etapa da análise de pontos de função consiste em identificar cinco componentes básicos, identificados como **arquivos lógicos internos (ALI)**, **arquivos de interface externa (AIE)**, **entrada externa (EE)**, **saída externa (SE)** e **consulta externa (CE)**, que por sua vez são enquadrados em duas categorias denominadas *funções do tipo dados* e *funções do tipo transação*.

- **Funções tipo dados**: conforme destacado por Lopes (2011, p. 9-10), as funções do tipo dados referem-se às funcionalidades fornecidas para o armazenamento de dados da aplicação e são caracterizadas como arquivos lógicos, podendo ser mantidas dentro ou fora da aplicação. Arquivos lógicos mantidos dentro da fronteira da aplicação são denominados *ALI*; já os arquivos lógicos mantidos fora da aplicação ou lidos de outra são chamados de *AIE*. Lopes (2011, p. 10) apresenta os seguintes exemplos:

a. **exemplos de ALI**: arquivos de configuração mantidos pela aplicação; tabelas ou grupos de tabelas do banco de dados mantidos pela aplicação;

b. **exemplos de AIE**: dados de segurança armazenados em arquivos lógicos e mantidos por aplicações específicas a esse fim; dados salariais armazenados em aplicação financeira, mas utilizados pela aplicação contada.

Parafraseando Silva e Oliveira (2005, p. 5), o processo de contagem consiste em identificar os registros lógicos referenciados (RLR) e seus dados lógicos referenciados (DER) para cada ALI ou AIE e, com base nisso, atribuir um grau de complexidade e um correspondente número de pontos de função. Um DER corresponde a um campo único, reconhecido pelo usuário, não repetido, por exemplo um atributo de uma tabela do banco de dados. Já um RLR é um subgrupo de DER, reconhecido pelo usuário, por exemplo uma tabela do banco de dados. Exemplificando, vamos supor um registro de clientes que possua uma tabela no banco de dados denominada *Clientes*, com os seguintes atributos: *código*, *nome*, *endereço*, *telefone*, *RG* e *CPF*. Nesse caso, temos um ALI que podemos denominar de *Clientes*, com um RLR (tabela *Clientes*) e seis DER (*código de atributos*, *nome*, *endereço*, *telefone*, *RG*, *CPF*). Vale salientar que chaves estrangeiras não são contadas como DER.

Os Quadros 1 e 2 apresentam a complexidade e a pontuação das funções tipo dados.

Quadro 1 – Complexidade das funções tipo dados

	1 a 19 DER	**20 a 50 DER**	**Acima de 50 DER**
1 RLR	Simples	Simples	Média
2 a 5 RLR	Simples	Média	Complexa
Acima de 5 RLR	Média	Complexa	Complexa

Fonte: Elaborado com base em Silva; Oliveira, 2005, p. 5.

Quadro 2 – Pontos de função das funções tipo dados

Função	**Simples**	**Média**	**Complexa**
ALI	7 pontos	10 pontos	15 pontos
AIE	5 pontos	7 pontos	10 pontos

Fonte: Elaborado com base em Silva; Oliveira, 2005, p. 5.

Exemplificando, suponhamos que, em um projeto de melhoria de uma aplicação, é adicionada a uma aplicação um processo de vendas. Para isso, são criadas as tabelas *Vendas* e *Itens_Venda*, no banco de dados. A tabela *Vendas* contém os atributos *código*, *cliente* e *data*; já a tabela *Itens_Venda* contém os atributos *código*, *produto*, *quantidade*, *valor*, *total*. Como sabemos que não podemos inserir uma venda na tabela *Vendas* sem itens correspondentes na tabela *Itens_Venda* e também não podemos realizar o processo inverso, essas duas tabelas correspondem a um ALI com dois RLR (tabelas *Vendas* e *Itens_Venda*) e seis DER (*código* e *data*, da tabela *Vendas*; e *código*, *quantidade*, *valor* e *total*, da tabela *Itens_Venda*). Os atributos *cliente*, da tabela *Vendas*, e *produto*, da tabela *Itens_Venda*, não são contados como DER, por serem chaves estrangeiras para outras tabelas. O Quadro 3 resume esses dados.

Quadro 3 – Exemplificação da contagem de pontos de função

Descrição	Tipo	RLR	DER	Complexidade	Pontos de função
Vendas	1 ALI	2 (Vendas e Itens_Venda)	6	Simples	7

De acordo com o Quadro 1, se temos um ALI com 2 a 5 RLR e até19 DER, a complexidade desse ALI é simples. Portanto, conforme apontado no Quadro 3, nosso ALI contém 2 RLR e 6 DER, o que o caracteriza com complexidade simples, nesse projeto de melhoria de aplicação. Portanto, analisando o Quadro 2, temos que cada ALI que possua complexidade simples recebe 7 pontos. Nesse caso, nosso ALI *clientes*, que representa a nova funcionalidade a ser implementada no projeto de melhoria, receberá um total de 7 pontos de função tipo dados.

- **Funções tipo transação**: conforme destacado por Silva e Oliveira (2005, p. 5), as funções do tipo transação representam as funcionalidades de processamento de dados do sistema, sendo classificadas em EE, SE e CE. Nesse contexto, Lopes (2011, p. 15) reforça que essas funções são denominadas como *processos elementares*, sendo que um processo elementar é a menor unidade de uma função disponível ao usuário. Nesse sentido, Lopes (2020, p. 15) ainda destaca que:

> Por exemplo, consultar clientes pode ser entendido como uma função, mas o mesmo não pode ser entendido como um processo elementar, uma vez que podem ser realizadas inúmeras consultas

diferentes aos clientes, como consultar clientes pelo nome, consultar clientes em débito, consultar registro de clientes e outras. Podemos perceber que cada consulta é uma funcionalidade única e independente, desta forma para determinar um processo elementar é necessário identificar todas as funcionalidades únicas e independentes de uma função. Assim, um processo elementar deve ser único. Por exemplo, consultas que diferem uma da outra pela organização dos dados gerados, não podem ser consideradas diferentes.

EE é um processo elementar que, conforme Vazquez, Simões e Albert (2009, p. 112), manipula dados ou informações de controle originados fora da fronteira de uma aplicação e tem como principal intenção manter (incluir, alterar ou excluir dados) um ou mais ALI e/ou alterar a forma como o sistema se comporta.

SE é um processo elementar que, segundo Vazquez, Simões e Albert (2009, p. 113), envia dados ou informações de controle para fora da fronteira da aplicação. Tem como principal intenção apresentar informação ao usuário, por meio de lógica de processamento, que não seja apenas a recuperação de dados ou informações de controle.

CE é um processo elementar que, de acordo com Vazquez, Simões e Albert (2009, p. 114), assim como a SE, envia dados ou informações de controle para fora da fronteira da aplicação. Tem como principal intenção apresentar informação ao usuário por meio de uma simples recuperação de dados ou informação de controle de ALI e/ou AIE. A sua lógica de processamento não deve conter fórmulas matemáticas ou cálculos.

O Quadro 4 apresenta alguns exemplos de EE, SE e CE.

Quadro 4 – Exemplos de EE, SE e CE

EE	SE	CE
Processos destinados a adicionar, excluir e alterar registros em ALI.	Relatórios que possuem totalização de dados.	Informações que possuem formato gráfico.

Fonte: Elaborado com base em Vazquez; Simões; Albert, 2009, p. 112-113.

Seguindo essa abordagem, para a contagem dos pontos de função tipo transação Silva e Oliveira (2005, p. 6) apontam que o processo consiste em identificar o ALR e seus DER para cada EE, SE ou CE e, com base nisso, atribuir-lhe(s) um grau de complexidade e um correspondente número de

pontos de função. O Quadro 5 apresenta o grau de complexidade das EE, o Quadro 6 apresenta o grau de complexidade das SE e CE e o Quadro 7 apresenta a pontuação das funções tipo transação.

Quadro 5 – Complexidade das EE

	1 a 4 DER	**5 a 15 DER**	**Acima de 15 DER**
1 ALR	Simples	Simples	Média
2 ALR	Simples	Média	Complexa
Acima de 2 ALR	Média	Complexa	Complexa

Fonte: Elaborado com base em Silva; Oliveira, 2005, p. 5.

Quadro 6 – Complexidade das SE e CE

	1 a 4 DER	**5 a 19 DER**	**Acima de 19 DER**
1 ALR	Simples	Simples	Média
2 a 3 ALR	Simples	Média	Complexa
Acima de 3 ALR	Média	Complexa	Complexa

Fonte: Elaborado com base em Silva; Oliveira, 2005, p. 5.

Quadro 7 – Pontos de função das funções tipo transação

Função	**Simples**	**Média**	**Complexa**
EE	3 pontos	4 pontos	6 pontos
SE	4 pontos	5 pontos	7 pontos
CE	3 pontos	4 pontos	6 pontos

Fonte: Elaborado com base em Silva; Oliveira, 2005, p. 5.

Exemplificando, vamos supor que, para o nosso projeto de melhoria de uma aplicação, foi necessária a criação dos processos elementares *incluir, alterar e excluir vendas*, *consultar clientes por idade*, *consultar total de compras por clientes* e *consultar produtos por ordem de preços*. O próximo

passo consiste na identificação de cada processo elementar em EE, SE e CE, conforme podemos observar no Quadro 8.

Quadro 8 – Definição dos processos elementares

Descrição	Tipo	ALR	DER	Complexidade
Incluir vendas	EE	1	8	Simples
Alterar vendas	EE	1	8	Simples
Excluir vendas	EE	1	8	Simples
Consultar clientes por idade	CE	1	9	Simples
Consultar produtos por ordem de preços	CE	1	6	Simples
Consultar total de compras por cliente	SE	3	12	Média

De acordo com o Quadro 8, obtivemos três EE com complexidade simples, duas CE com complexidade simples e uma SE com complexidade média. Portanto, considerando a complexidade de cada função, é possível verificar, no Quadro 7, a pontuação correspondente a cada função, que deve ser multiplicada pela quantidade de funções daquela complexidade. Sendo assim, temos:

- 3 EE simples x 3 = 9 pontos de função;
- 2 CE simples x 3 = 6 pontos de função;
- 1 SE média x 5 = 5 pontos de função;
- total de 20 pontos de função tipo transação.

1.4 DETERMINAR A CONTAGEM DE PONTOS DE FUNÇÃO NÃO AJUSTADOS

Após obtermos o total de pontos de função das funções tipo dados e das funções tipo transação, basta somarmos os dois resultados para obtermos o total de pontos de função não ajustados, que representam o tamanho funcional da aplicação. Analisando o nosso exemplo, como apresentado no Quadro 3, somando-se os 7 pontos de função que obtivemos como total de pontos de função tipo dados mais os 20 pontos que obtivemos como total de pontos de função tipo transação, obtemos um

total de 27 pontos de função não ajustados, ou seja, que representam o tamanho funcional das implementações realizadas em nosso projeto de melhoria de aplicação.

1.5 DETERMINAR O VALOR DO FATOR DE AJUSTE

Conforme abordado por Vazquez, Simões e Albert (2009), como as características intrínsecas a um aplicativo afetam o seu tamanho funcional, a metodologia introduz a avaliação de 14 características, conhecidas como *itens de influência*. Cada um desses itens recebe uma nota de 0 a 5, conforme análise do analista do projeto, podendo variar o tamanho funcional do software em até 35%, para mais ou para menos. Esse fator, aplicado aos pontos não ajustados, nos permite obter os chamados *pontos ajustados da aplicação*. Conforme Silva e Oliveira (2005, p. 6), o uso do fator de ajuste tornou-se opcional no final de 2002, dado que várias características estão desatualizadas ou sujeitas a interpretações diversas. O Quadro 9 apresenta os 14 itens de influência.

Quadro 9 – Itens de influência

1. Comunicação de dados	2. Funções distribuídas
3. Performance	4. Utilização do equipamento
5. Volume de transações	6. Entrada de dados *on-line*
7. Interface com o usuário	8. Atualizações *on-line*
9. Processamento complexo	10. Reusabilidade
11. Facilidade de implantação	12. Facilidade operacional
13. Múltiplos locais	14. Facilidade de mudanças

Fonte: Elaborado com base em Silva; Oliveira, 2005, p. 7.

Com base nos itens de influência apresentados no Quadro 9 e sabendo-se que cada item pode receber uma pontuação de 0 a 5, o fator de ajuste é derivado da aplicação da seguinte fórmula:

Fator de ajuste = (pontos de influência x 0,01) + 0,65.

Exemplificando isso em nossa abordagem anterior, vamos supor que consideramos as seguintes pontuações nos itens de influência:

- funções distribuídas – 2 pontos;
- interface com o usuário – 5 pontos;
- reusabilidade – 3 pontos;
- facilidade de implantação – 3 pontos;
- múltiplos locais – 4 pontos.

Nesse caso, temos um total de 17 pontos de influência, que, aplicados na fórmula, obtemos o seguinte fator de ajuste:

$$\mathbf{Fa = (17 \times 0{,}01) + 0{,}65};$$

$$\mathbf{Fa = 0{,}82}.$$

1.6 CALCULAR O NÚMERO DOS PONTOS DE FUNÇÃO AJUSTADOS (AFP)

De posse do total de pontos não ajustados e do fator de ajuste obtido após a definição dos itens de influência, basta multiplicarmos um pelo outro para obtermos o total de AFP. Voltando ao nosso exemplo anterior, como obtivemos um total de 27 pontos de função não ajustados e um fator de ajuste no valor de 0,82, ao multiplicarmos um pelo outro obtemos um total de 22,14 AFP, que correspondem ao tamanho funcional da nossa aplicação. Porém, devemos nos recordar de que esse é um processo opcional. Portanto, não é necessária a definição dos itens de influência. Poderíamos, perfeitamente, considerar os 27 pontos da nossa contagem anterior como AFP.

1.7 DURAÇÃO E CUSTO DE UM PROJETO

Após obter o tamanho funcional do software, ou seja, o total de AFP, para calcular o esforço (*E*) total do desenvolvimento é necessário saber quantos pontos de função são produzidos por hora e saber quantas horas de trabalho são consideradas por mês na empresa.

De acordo com Lopes (2011, p. 26), cada linguagem ou tecnologia demanda um esforço diferente e suas características não influenciam nos pontos de função, mas sim nesse esforço que demanda produzir cada ponto de função. Desse modo, poderíamos considerar como critério de exemplo que um projeto em Java pode apresentar até 15 horas de desenvolvimento por ponto de função, enquanto um projeto em Hypertext Preprocessor (PHP) poderia levar 11 horas por ponto de função.

Diante dessa abordagem, Lopes (2011, p. 26) destaca que, para se definir o tempo de desenvolvimento de um projeto, em horas, por ponto de função, o mais importante é se possuir um histórico de projetos, de modo a se estimar esse esforço de desenvolvimento com maior precisão. Na primeira vez que essas estimativas forem aplicadas, o erro poderá será grande, mas, conforme a base de históricos de projetos for sendo ampliada, o erro tenderá a diminuir.

Para efeito de exemplificação do esforço necessário para o desenvolvimento da nossa aplicação, que foi estimada em 22,14 pontos de função, consideremos que esse projeto será desenvolvido em Python e que possuímos uma taxa de produtividade mínima, nessa linguagem, de 7 horas por ponto de função (H/PF) e uma carga de trabalho de 160 horas por desenvolvedor ao mês (considerando-se uma carga de trabalho de 8 horas diárias e 5 dias por semana). Portanto, temos que:

$$\text{Esforço} = 7 \times 22{,}14 = 154{,}98 \text{ horas.}$$

Conforme nosso exemplo, precisamos de aproximadamente 155 horas para produzir o sistema ou quase um mês de trabalho de um desenvolvedor.

Em relação ao custo do projeto, Lopes (2011, p. 27) aponta que, como na determinação do esforço, o custo também é estimado com base em dados da empresa que irá desenvolver o projeto. Nesse caso, pois, é necessário ter o conhecimento do custo da hora da equipe de desenvolvimento ou o valor de um ponto de função para a empresa.

Seguindo essa abordagem, Wazlawick (2013, p. 165) destaca que o custo do projeto é calculado como o esforço total multiplicado pelo custo médio da hora do desenvolvedor, conforme segue:

$$\text{Custo} = E * \text{Custo}_{\text{horaDev}}.$$

Exemplificando, vamos supor que o custo da hora de trabalho de um desenvolvedor em Python seja R$ 27,00 e, como será produzido um ponto de função a cada 7 horas, o valor do ponto de função seja de:

$$7 \text{ horas} \times \text{R\$ } 27{,}00 = \text{R\$ } 189{,}00.$$

Desse modo, como os esforços necessários para produzir essa aplicação em Python são de 155 horas e ela possui 22,14 pontos de função, o custo será:

$$\text{Custo} = 22{,}14 \times \text{R\$ } 189{,}00 = \text{R\$ } 4.184{,}46$$

ou

Custo = 154,98 horas x R$ 27,00 = R$ 4.184,46.

Outras estimativas, como o cálculo do tempo linear de desenvolvimento e o tamanho médio da equipe, serão abordadas no Tema 5 desta aula, quando discutirmos sobre o modelo Cocomo, pois os cálculos utilizados são os mesmos para todas as métricas estudadas nesta aula.

TEMA 2 – PONTOS DE CASOS DE USO

A técnica de análise de pontos de casos de uso surgiu em 1993 como um modelo relativamente mais simples do que a análise de pontos de função. Conforme Wazlawick (2013, p. 171): "O método se baseia na análise da quantidade e complexidade dos atores e casos de uso, o que gera os UUCP, ou pontos de caso de uso não ajustados. Depois, a aplicação dos fatores técnicos e ambientais levam aos UCP, ou pontos de caso de uso ajustados.".

A contagem dos pontos de casos de uso inicia-se pela definição da complexidade dos atores dos casos de uso. Conforme Wazlawick (2013, p. 171), cada ator é contado uma única vez, mesmo que esteja relacionado a vários casos de uso. Identificados os atores, a complexidade é definida conforme segue:

- Atores humanos que interagem com o sistema por meio de interface gráfica: alta complexidade – 3 pontos de caso de uso.
- Sistemas que interagem por protocolo de controle de transmissão/protocolo de internet (TCP/IP) e atores humanos que interagem com o sistema apenas por linha de comando: média complexidade – 2 pontos de caso de uso.
- Sistemas que são acessados por interfaces de programação (API): baixa complexidade – 1 ponto de caso de uso.

Após a identificação e pontuação dos atores, basta somente realizar a soma de todos os valores para obter-se o peso não ajustado dos atores. Da mesma forma que os atores, os casos de uso também devem ser contados uma única vez e somente casos de uso completos devem ser contados. Conforme apresentado por Wazlawick (2013, p. 171), existem três formas de definição de complexidade dos casos de uso, conforme seguem:

1. Originalmente, a complexidade de um caso de uso é definida em função do número estimado de transações, incluindo as sequências alternativas do caso de uso, cuja classificação é em:

a. casos de uso simples: com até 3 transações, valem 5 pontos;

b. casos de uso médios: com 4 a 7 transações, valem 10 pontos;

c. casos de uso complexos: com acima de 7 transações, valem 15 pontos.

2. Outra forma de se estimar a complexidade de um caso de uso é em função da quantidade de classes necessárias para implementar as funções do caso de uso, cuja classificação abrange:

a. casos de uso simples: com 5 classes ou menos.

b. casos de uso médios: com 6 a 10 classes;

c. casos de uso complexos: com mais de 10 classes.

3. A última forma de se estimar a complexidade de um caso de uso é pela análise de seu risco, pela qual há:

a. casos de uso simples: casos de uso como relatórios que têm apenas 1 ou 2 transações e baixo risco, pois não alteram dados.

b. casos de uso médios: casos de uso padronizados que têm um número conhecido e limitado de transações, pois, embora a sua lógica de funcionamento seja conhecida, neles, regras de negócio obscuras podem existir;

c. casos de uso complexos: casos de uso não padronizados que têm um número desconhecido de transações e alto risco, pois, além de as regras de negócio serem desconhecidas, ainda deve-se descobrir qual é o fluxo principal e quais as sequências alternativas.

Para obter-se o total dos pontos de casos de uso não ajustados (UUCP), basta somar todos os pontos atribuídos aos casos de uso. Finalmente, para o cálculo dos UUCP, basta que somemos o total do peso não ajustado dos atores com o peso não ajustado dos casos de uso.

2.1 PONTOS AJUSTADOS DE CASOS DE USO

Conforme ocorre na análise de pontos de função com a aplicação dos itens de influência, na análise de pontos de casos de uso para a definição dos pontos ajustados são aplicados 13 fatores técnicos, que também recebem uma pontuação de 0 a 5; porém, ao contrário da análise de pontos de função, cada fator técnico recebe inicialmente um valor predefinido. O Quadro 10 apresenta os 13 fatores técnicos e seus pesos iniciais.

Quadro 10 – Fatores técnicos da análise de pontos de casos de uso

Fator técnico	Peso	Fator técnico	Peso
Sistema distribuído	2	*Performance*	2
Eficiência de usuário final	1	Complexidade de processamento	1
Projeto visando a código reusável	1	Facilidade de instalação	0,5
Facilidade de uso	0,5	Portabilidade	2
Facilidade de mudança	1	Concorrência	1
Segurança	1	Acesso fornecido a terceiros	1
Necessidade de treinamento	1		

Fonte: Elaborado com base em Wazlawick, 2013, p. 172.

Após a definição da pontuação dos fatores técnicos pelo analista do sistema, o fator de complexidade técnica (TCF) é calculado conforme a seguinte fórmula:

TCF = 0,6 + (0,01 * total da soma dos pontos de fatores técnicos).

Além dos 13 fatores técnicos, também são aplicados mais 8 fatores ambientais, que são específicos às características da equipe de desenvolvimento. Parafraseando Wazlawick (2013, p. 173), desse modo um mesmo projeto, com os mesmos fatores técnicos, poderá ter pontos de caso de uso ajustados distintos, para equipes diferentes. O Quadro 11 apresenta os 8 fatores ambientais e seus respectivos pesos iniciais.

Quadro 11 – Fatores ambientais da análise de pontos de casos de uso

Fator ambiental	Peso	Fator ambiental	Peso

Familiaridade com o processo de desenvolvimento	1,5	Experiência com a aplicação	0,5
Experiência com orientação a objetos	1	Capacidade do analista líder	0,5
Motivação	1	Estabilidade de requisitos obtida historicamente	2
Equipe em tempo parcial	-1	Dificuldade com a linguagem de programação	-1

Fonte: Elaborado com base em Wazlawick, 2013, p. 173.

Similarmente ao TCF, o fator ambiental total (EF) é calculado com base na fórmula que segue:

EF = 1,4 – (0,03 x total da soma dos fatores ambientais).

Finalizando, para obtermos o total de pontos de casos de uso ajustados (UCP), basta multiplicarmos o total de UUCP por TCF e EF, conforme segue:

UCP = UUCP * TCF * EF.

TEMA 3 – PONTOS DE HISTÓRIAS

Pontos de histórias são uma técnica de estimativa de software variante da análise de pontos de função. Conforme abordados por Wazlawick (2013, p. 177), são a estimativa de esforço preferida de métodos ágeis como Scrum e XP. Embora caiba ressalvar, segundo Wazlawick (2013, p. 177), que: "Um ponto de história não é uma medida de complexidade funcional, como os pontos de função ou pontos de casos de uso, mas uma medida de esforço relativa à equipe de desenvolvimento."

Vale salientar aqui que uma história de usuário corresponde a uma explicação informal e geral sobre um recurso de software, escrita sob a perspectiva do usuário final. Portanto, conforme aborda Kniberg (2007), a metodologia consiste em identificar de forma subjetiva, com a equipe de desenvolvimento, quanto tempo um número qualquer de pessoas que se dedicasse unicamente a uma história de usuário levaria para terminá-la, com uma versão executável do sistema. Com base nessas respostas, multiplica-se o número de pessoas pela quantidade de dias para se obter o total de pontos de história. Por exemplo, se 2 pessoas levariam 6 dias para implementar determinada história de usuário, então atribuem-se à história: 2 x 6 = 12 pontos de história.

Conforme exposto por Wazlawick (2013, p. 177), uma alternativa utilizada nos métodos ágeis é atribuir pontos às histórias de acordo com o seu tempo de desenvolvimento, pois, "[...] do ponto de

vista da agilidade, é mais importante saber que uma história levaria duas vezes mais tempo do que outra para ser implementada, do que quantos dias efetivamente levaria para ser implementada."

Portanto, um método para trabalhar essa alternativa é atribuir pontos similares à série de Fibonacci, na qual a soma dos dois números anteriores define o terceiro, conforme segue: 1, 1, 2, 3, 5, 8, 13, 21, 34... Para facilitar a compreensão, esses valores podem ser arredondados ou atribuídos como múltiplos de determinado valor, conforme segue: 1, 2, 3, 5, 10, 15, 20, 35...

Dado o exposto, cada história recebe uma pontuação de acordo com o seu tempo de desenvolvimento: histórias rápidas de serem desenvolvidas recebem valores menores; e histórias que demandam mais tempo de desenvolvimento recebem valores maiores.

TEMA 4 – SLOC E KSLOC

Source lines of code (Sloc) corresponde ao número de linhas de código implementadas em um sistema. Essa técnica, conforme aborda Wazlawick (2013, p. 130), é possivelmente uma das primeiras técnicas de estimativas e consiste em estimar o número de linhas que um programa deverá ter, normalmente com base na opinião de especialistas e no histórico de projetos passados.

De acordo com Wazlawick (2013, p. 130):

> [...] a técnica evoluiu para a forma conhecida como KSLOC (Kilo Source Lines of Code), dado que o tamanho da maioria dos programas passou a ser medido em milhares de linhas. Uma unidade KSLOC vale mil unidades SLOC, ou seja, um software que apresente 10.000 SLOC é representado como 10 KSLOC. Além disso, também são usados os termos MSLOC para milhões de linhas e GSLOC para bilhões de linhas de código.

Uma técnica da estimação de *kilo source lines of code* (Ksloc), apresentada por Wazlawick (2013, p. 130), é se reunir a equipe de projeto para discutir o sistema a ser desenvolvido. Cada participante dá a sua opinião sobre a quantidade de Ksloc que será necessária para desenvolver o sistema. Como em geral a equipe não chega a um valor único, devem ser considerados pelo menos três valores:

1. **Ksloc otimista**: número mínimo de linhas que se espera desenvolver se todas as condições forem favoráveis.
2. **Ksloc pessimista**: número máximo de linhas que se espera desenvolver em condições desfavoráveis.

3. **Ksloc esperado**: número de linhas que efetivamente se espera desenvolver em uma situação de normalidade.

Com base nesses valores, o Ksloc é calculado conforme esta fórmula:

$$\mathbf{KSLOC = (4 \times KSLOC_{esperado} + KSLOC_{otimista} + KSLOC_{pessimista}) / 6}.$$

Exemplificando, vamos supor uma equipe de desenvolvimento com três membros em que, para desenvolver um novo projeto de software, cada membro sugere valores de Ksloc em milhares de linhas de código, conforme apresentado no Quadro 12.

Quadro 12 – Estimativa de Ksloc em milhares de linhas de código

	KSLOC otimista	KSLOC pessimista	KSLOC esperado
Membro 1	15	10	13
Membro 2	20	13	17
Membro 3	18	13	15
Média	17,66	12	15

Aplicando os valores apresentados no Quadro 12 na fórmula para calcular o Ksloc, temos a seguinte situação:

$$\mathbf{KSLOC = (4 \times KSLOC_{esperado} + KSLOC_{otimista} + KSLOC_{pessimista}) / 6};$$

$$\mathbf{KSLOC = (4 \times 15 + 17{,}66 + 12) / 6};$$

$$\mathbf{KSLOC = 14{,}94}.$$

Portanto, com base no resultado obtido, para o desenvolvimento do novo projeto de software será considerado que esse projeto terá 14.940 Ssloc ou 14,94 Ksloc.

Wazlawick (2013, p. 130) salienta que as estimativas obtidas devem ser comparadas com a informação real, ao final do projeto, para a equipe ter um feedback para ajustar futuras previsões. Assim, uma técnica que pode ser empregada para ajustar a capacidade de previsão da equipe é tomar por base projetos já desenvolvidos.

TEMA 5 – COCOMO

O modelo Cocomo, em sua versão mais atual conhecido como Cocomo II, é um modelo de estimativa de esforço para produção de software baseado em Ksloc. De acordo com Wazlawick (2013, p. 134), o modelo Cocomo apresenta-se em três formas de implementação, de acordo com a complexidade dessa implementação e o grau de informações que se tenha a respeito do sistema a ser desenvolvido:

1. **Implementação básica**: quando a única informação sobre o sistema efetivamente disponível é o número estimado de linhas de código.
2. **Implementação intermediária**: quando fatores relativos a produto, suporte computacional, pessoal e processo são conhecidos.
3. **Implementação avançada**: quando for necessário subdividir o sistema em subsistemas e distribuir as estimativas de esforço por fases e atividades.

Além dos três modos de implementação, conforme abordado por Wazlawick (2013, p. 134), para o cálculo do esforço todas as implementações consideram também o tipo de projeto a ser desenvolvido, que pode ser:

a. **Modo orgânico**: aplica-se quando o sistema a ser desenvolvido é de baixa complexidade e a equipe está acostumada a desenvolver esse tipo de aplicação; de baixo risco tecnológico; e de baixo risco de pessoal.

b. **Modo semidestacado**: aplica-se a sistemas com maior grau de novidade para a equipe, sistemas nos quais a combinação do risco tecnológico e de pessoal seja média.

c. **Modo embutido**: aplica-se a sistemas com alto grau de complexidade ou que sejam embarcados, para os quais a equipe tenha considerável dificuldade de abordagem. São os sistemas com alto risco tecnológico e/ou de pessoal.

Conforme destaca Wazlawick (2013, p. 134), as três formas de implementação permitem determinar três informações básicas:

1. o esforço estimado em desenvolvedor/mês, dado pela notação *E*;
2. o tempo linear de desenvolvimento, sugerido em meses corridos, dado pela notação *T*;
3. o número médio de pessoas recomendado para a equipe, dado pela notação *P*.

Constituem **estimativas de esforço no modelo básico**:

- esforço estimado em desenvolvedor/mês:

$$E = ax * KSLOC^{bx},$$

em que os valores *ax* e *bx* são constantes e estabelecidas de acordo com o tipo de projeto desenvolvido, conforme mostrado no Quadro 13. Já o número dos milhares de linhas de código (Ksloc) será obtido por meio das estimativas de Ksloc otimista**,** Ksloc pessimista e Ksloc esperado definidas pela equipe de desenvolvimento, conforme abordado no Tema 4.

Quadro 13 – Constantes *a* e *b* em função do tipo de projeto

Tipo	ax	bx	cx	dx
Orgânico	2,4	1,05	2,5	0,38
Semidestacado	3	1,12	2,5	0,35
Embutido	3,6	1,2	2,5	0,32

Fonte: Elaborado com base em Wazlawick, 2013, p. 135.

- tempo linear de desenvolvimento:

$$T = cx * E^{dx}.$$

- número médio de pessoas recomendado para a equipe:

$$P = E / T.$$

Quanto às **estimativas de esforço nos modelos intermediário e avançado**, o modelo intermediário considera, além das linhas de código, 15 fatores influenciadores de custo, que, como vimos anteriormente, são relativos a produto, suporte computacional, pessoal e processo (Wazlawick, 2013, p. 136). O Quadro 14 apresenta esses fatores e seus respectivos valores.

Quadro 14 – Fatores influenciadores de custo, conforme acrônimo e complexidade de implementação

Fatores influenciadores de custo	Acrônimo	Muito baixa	Baixa	Média	Alta	Muito alta	Extra-alta
Relativos ao produto							
Nível de confiabilidade requerida	Rely	0,75	0,88	1	1,15	1,4	
Dimensão da base de dados	Data		0,94	1	1,08	1,16	
Complexidade do produto	CPLX	0,7	0,85	1	1,15	1,3	1,65
Suporte computacional							
Restrições ao tempo de execução	Time			1	1,11	1,3	1,66
Restrições ao espaço de armazenamento	Stor		0	1	1,06	1,21	1,56
Volatilidade da máquina virtual	Virt		0,87	1	1,15	1,3	
Tempo de resposta do computador	Turn		0,87	1	1,07	1,15	
Pessoal							
Capacidade dos analistas	Acap	1,46	1,19	1	0,86	0,71	
Experiência no domínio da aplicação	Aexp	1,29	1,13	1	0,91	0,72	
Capacidade dos programadores	Pcap	1,42	1,17	1	0,86	0,7	
Experiência na utilização da máquina virtual	Vext	1,21	1,1	1	0,9		
Experiência na linguagem de programação	Lexp	1,14	1,07	1	0,95		
Processo							
Adoção de boas práticas de programação	Modp	1,24	1,1	1	0,91	0,82	
Uso de ferramentas atualizadas	Tool	1,24	1,1	1	0,91	0,82	
Histórico de projetos terminados no prazo	Sced	1,23	1,08	1	1,04	1,1	

Fonte: Elaborado com base em Wazlawick, 2013, p. 136.

Conforme exposto por Wazlawick (2013, p. 136), os 15 fatores influenciadores de custo são combinados por meio de multiplicação, produzindo o valor:

$$EAF = Rely * Data * CPLX * Time * ... * Sced.$$

Portanto, a estimativa de esforço é dada por:

$$E = ai * Ksloc^{bi} * EAF.$$

Os valores *ai* e *bi* são constantes e estabelecidas de acordo com o tipo de projeto desenvolvido, conforme mostrado no Quadro 15.

Quadro 15 – Constantes *ai* e *bi* em função do tipo de projeto

Tipo	ai	bi
Orgânico	2,8	1,05
Semidestacado	3	1,12
Embutido	3,2	1,2

Fonte: Wazlawick, 2013, p. 137.

As outras duas estimativas, referentes ao tempo linear de desenvolvimento e ao número médio de pessoas recomendadas para a equipe, seguem o mesmo padrão do modelo orgânico.

FINALIZANDO

Em nossa quarta aula, estudamos como realizar estimativas de esforço para desenvolvimento de softwares. Abordamos, nesta aula, como estimar as quatro principais variáveis condicionantes de um projeto, sendo elas o esforço necessário para o desenvolvimento do projeto de software, o tempo linear de desenvolvimento, o tamanho adequado da equipe de desenvolvimento e o custo de desenvolvimento do projeto.

No decorrer da aula, analisamos ainda cinco tipos diferentes de métricas que podem ser aplicadas para estimar as quatro variáveis identificadas. No Tema 1 estudamos a técnica de medição de análise de pontos de função, uma das mais conhecidas e possivelmente a técnica mais utilizada. No Tema 2 aprendemos a técnica de pontos de casos de uso e, no Tema 3, a técnica de pontos de história, ambas sendo variantes da técnica de análise de pontos de função e também mais simples de serem aplicadas que aquela. Finalizamos esta aula abordando, no Tema 4, as técnicas mais rudimentares Sloc e Ksloc, que realizam estimativas baseadas no número de linhas de código do programa; e, no Tema 5, o modelo Cocomo, que auxilia todas as técnicas estudadas, com base nos valores Ksloc, para determinar principalmente o tempo linear e o tamanho adequado da equipe de desenvolvimento de projeto.

Recapitulando, estudamos os tipos de modelos de processos prescritivos de software e os métodos ágeis; na sequência, examinamos como utilizá-los na arquitetura de um projeto de software, assim como os tipos de arquitetura, até determinarmos custo, equipe e esforço para o desenvolvimento nesta aula. Portanto, na próxima aula aprenderemos os conteúdos do teste de software, que pode ser aplicado tanto na fase de desenvolvimento quanto em um projeto concluído, focando principalmente em testes de estrutura e funcionalidade.

REFERÊNCIAS

KNIBERG, H. **Scrum and XP from the Trenches**: How We Do Scrum. [S.l.]: InfoQ, 2007.

LOPES, J. S. **Guia prático em análise de ponto de função**. Viçosa: UFV, 2011. Disponível em: <https://www.fattocs.com/wp-content/uploads/2020/04/JhoneySLopes-JoseLBraga-2011.pdf>. Acesso em: 17 fev. 2021.

SILVA, I. J. M.; OLIVEIRA, V. **Qualidade em software**: uma metodologia para homologação de sistemas. Rio de Janeiro: Alta Books, 2005.

VAZQUEZ, C. E.; SIMÕES, G. S.; ALBERT, R. M. **Análise de ponto de função**: medição, estimativa e gerenciamento de projetos de software. São Paulo: Érica, 2009.

WAZLAWICK, R. S. **Engenharia de software**: conceitos e práticas. São Paulo: Elsevier, 2013.